

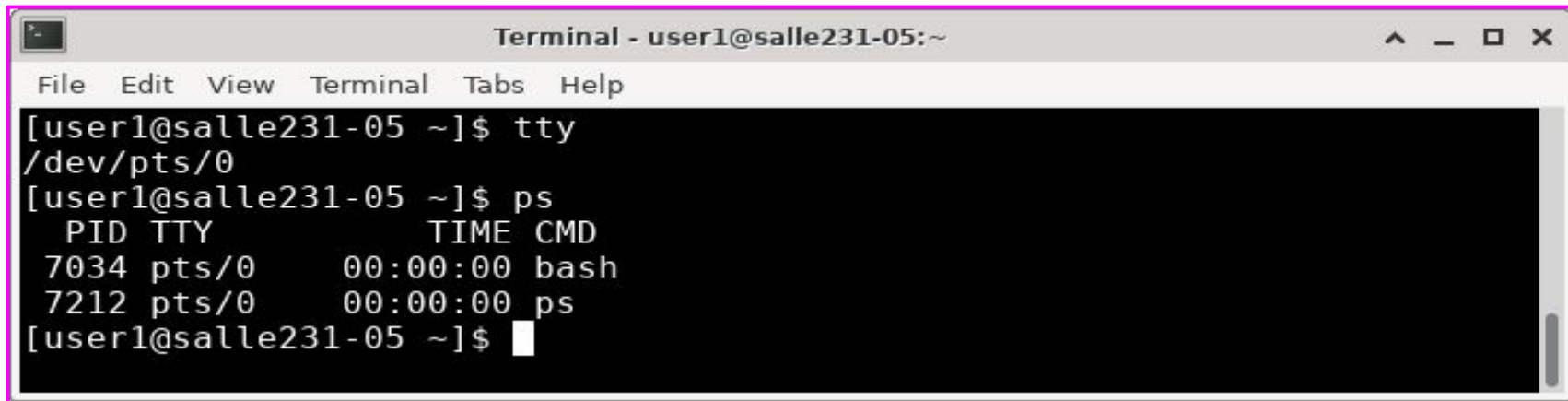
Scripts-shell

Selma - Département Informatique - IUT Fontainebleau/Sénart

➡ Commande externe, commande interne.

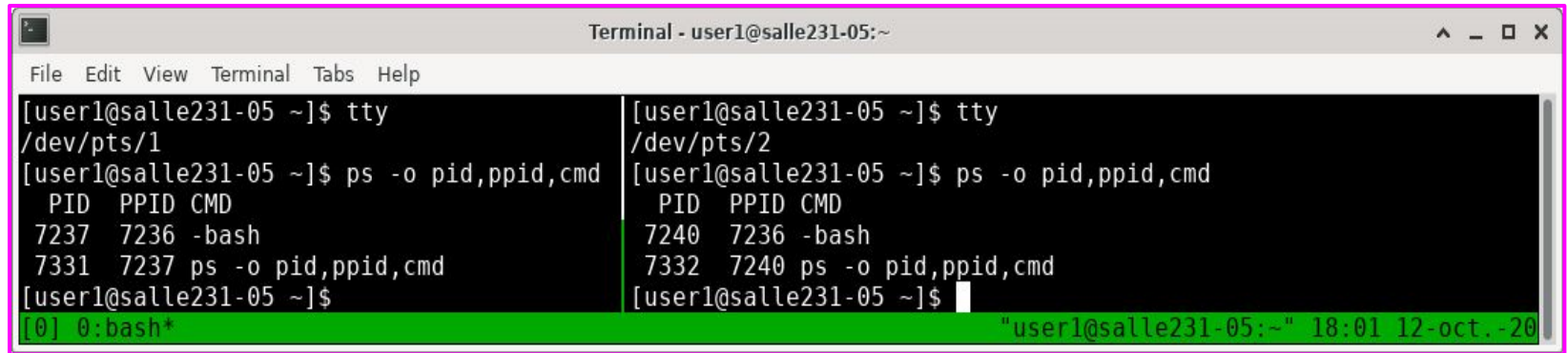
Quasiment toutes les commandes du shell sont externes, c'est à dire, pour les exécuter le shell demande (au noyau) la création d'un processus shell et son recouvrement par le code exécutable du binaire correspondant à la commande. Par opposition, une commande interne au shell est donc une commande dont le code exécutable est connu du shell et qu'il peut exécuter lui-même directement.

Dans les exemples suivants, on illustre le propos. On commence par consulter des informations sur le bash où on va soumettre des lignes de commandes...

A terminal window titled "Terminal - user1@salle231-05:~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following commands and output:

```
[user1@salle231-05 ~]$ tty
/dev/pts/0
[user1@salle231-05 ~]$ ps
  PID TTY          TIME CMD
 7034 pts/0        00:00:00 bash
 7212 pts/0        00:00:00 ps
[user1@salle231-05 ~]$
```

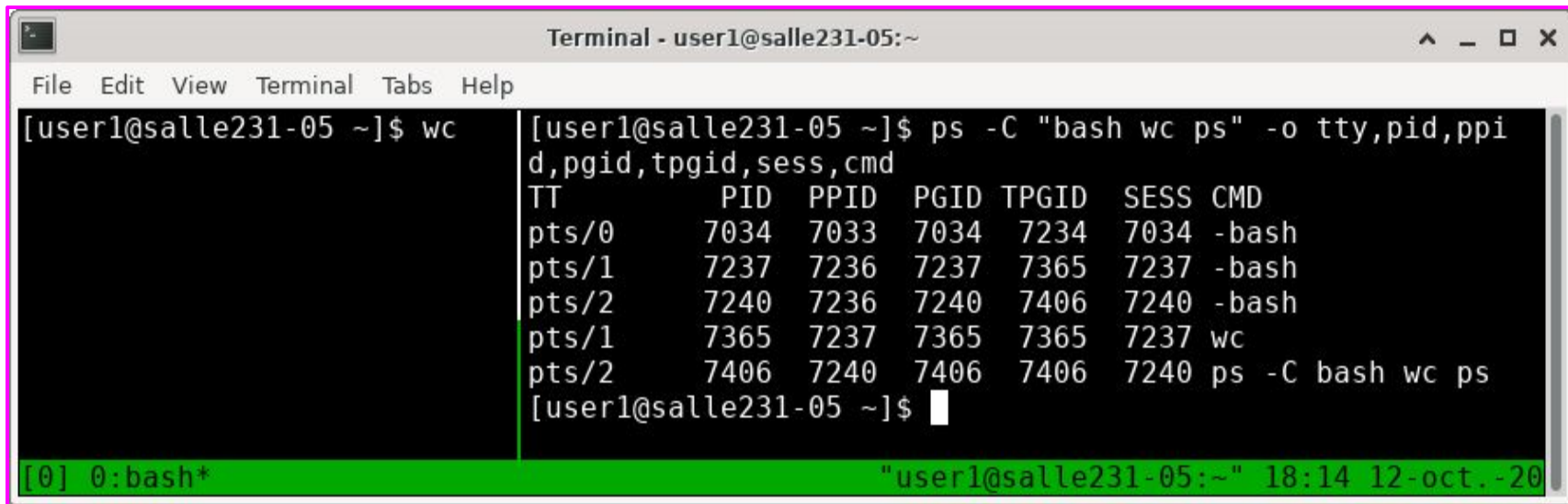
→ Une commande externe.



```
Terminal - user1@salle231-05:~  
File Edit View Terminal Tabs Help  
[user1@salle231-05 ~]$ tty  
/dev/pts/1  
[user1@salle231-05 ~]$ ps -o pid,ppid,cmd  
  PID  PPID  CMD  
 7237  7236  -bash  
 7331  7237  ps -o pid,ppid,cmd  
[user1@salle231-05 ~]$  
[0] 0:bash*  
[user1@salle231-05 ~]$ tty  
/dev/pts/2  
[user1@salle231-05 ~]$ ps -o pid,ppid,cmd  
  PID  PPID  CMD  
 7240  7236  -bash  
 7332  7240  ps -o pid,ppid,cmd  
[user1@salle231-05 ~]$  
"user1@salle231-05:~" 18:01 12-oct.-20
```

La commande ps a été exécutée par un processus autre que le bash où on a soumis la ligne de commande ps. Ainsi, ps est une commande externe au shell.

→ Encore une commande externe.

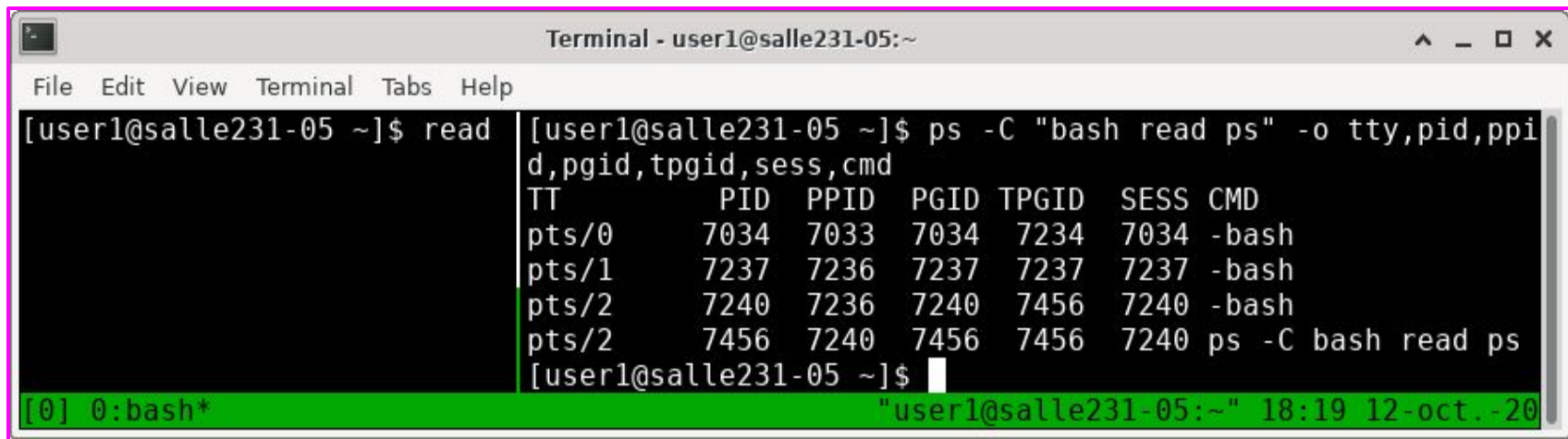


```
Terminal - user1@salle231-05:~
File Edit View Terminal Tabs Help
[user1@salle231-05 ~]$ wc
[user1@salle231-05 ~]$ ps -C "bash wc ps" -o tty,pid,ppid,pgid,tpgid,se ss,cmd
TT          PID  PPID  PGID  TPGID  SESS  CMD
pts/0       7034  7033  7034  7234   7034  -bash
pts/1       7237  7236  7237  7365   7237  -bash
pts/2       7240  7236  7240  7406   7240  -bash
pts/1       7365  7237  7365  7365   7237  wc
pts/2       7406  7240  7406  7406   7240  ps -C bash wc ps
[user1@salle231-05 ~]$
```

[0] 0: bash* "user1@salle231-05:~" 18:14 12-oct.-20

La commande wc est exécutée par un processus dont le parent est le bash où on a soumis la commande wc.

→ Une commande interne.



```
Terminal - user1@salle231-05:~
File Edit View Terminal Tabs Help
[user1@salle231-05 ~]$ read
[user1@salle231-05 ~]$ ps -C "bash read ps" -o tty,pid,ppid,pgid,tpgid,sess,cmd
TT      PID  PPID  PGID  TPGID  SESS  CMD
pts/0    7034  7033   7034   7234   7034  -bash
pts/1    7237  7236   7237   7237   7237  -bash
pts/2    7240  7236   7240   7456   7240  -bash
pts/2    7456  7240   7456   7456   7240  ps -C bash read ps
[user1@salle231-05 ~]$
```

[0] 0:bash* "user1@salle231-05:~" 18:19 12-oct.-20

La commande read est exécutée par le bash même où on soumit la commande read.

Un fichier contenant des lignes de commandes en shell

Lorsqu'une même séquence de plusieurs lignes de commandes doit être passée fréquemment, il est préférable d'en faire le contenu d'un fichier.

Par convention, on donne à un tel fichier un nom avec l'extension sh. Ceci permet de reconnaître facilement les fichiers dont le contenu est constitué de lignes de commandes shell.

On a deux ou trois façons de faire exécuter le contenu d'un tel fichier par le shell. On utilisera la manière suivante.

On rend le fichier exécutable, puis on le lance directement à la ligne de commandes par son nom ou par le chemin complet vers son nom selon que le répertoire qui le contient fasse partie ou pas de la liste qui constitue la valeur de la variable PATH.

Terminal - user1@salle231-05:~

File Edit View Terminal Tabs Help

```
#!/bin/bash
## A simple script shell which renames files
## in some given directory so that each name is
## followed by some given extension.
```

```
if [[ $# -lt 2 ]]
then
    echo "Usage: $0 <string> <DIR>"
    exit
fi
```

```
for n in `ls $2/*`
do
    mv $n $n.$1
done
```

```
exit
```

```
(END)
```

```
[user1@salle231-05 ~]$
```

```
[0] 0:less*
```

```
"user1@salle231-05:~" 19:04 12-oct.-20
```

```
Terminal - user1@salle231-05:~
File Edit View Terminal Tabs Help

[user1@salle231-05 ~]$ ls
AAA BBB ddd Desktop example.sh
[user1@salle231-05 ~]$ ls AAA BBB
AAA:
a b c

BBB:
s t x y z
[user1@salle231-05 ~]$

[user1@salle231-05 ~]$ ./example.sh
Usage: ./example.sh <string> <DIR>
[user1@salle231-05 ~]$ ./example.sh str
Usage: ./example.sh <string> <DIR>
[user1@salle231-05 ~]$ ./example.sh asr
Usage: ./example.sh <string> <DIR>
[user1@salle231-05 ~]$ ./example.sh asr ./AAA
[user1@salle231-05 ~]$ ls ./AAA ./BBB
./AAA:
a.asr b.asr c.asr

./BBB:
s t x y z
[user1@salle231-05 ~]$
```

[0] 0:bash* "user1@salle231-05:~" 19:09 12-oct.-20

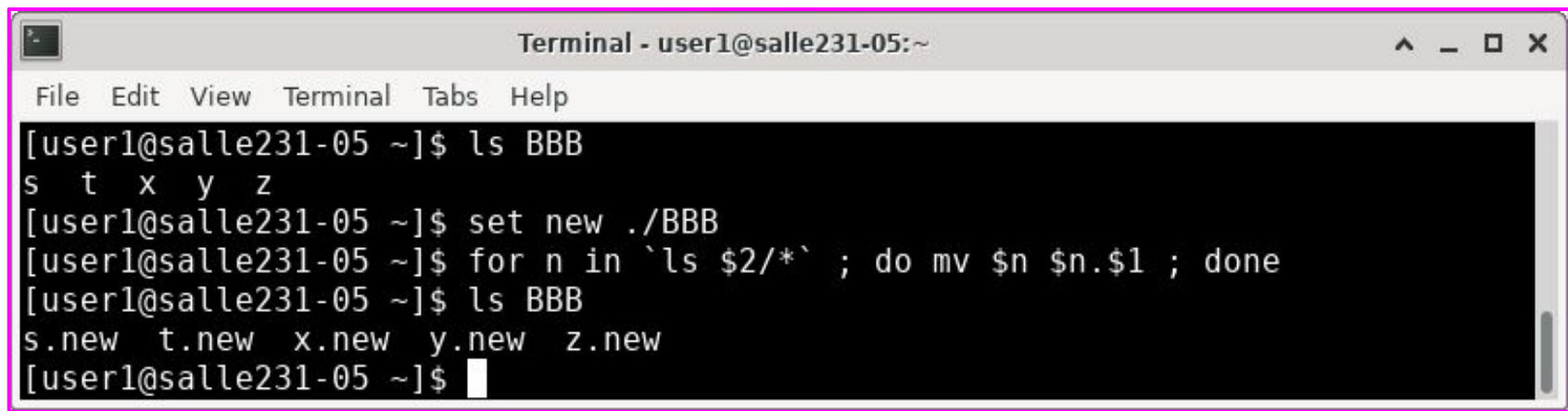
➡ Les paramètres positionnels du shell.

Un script-shell qui prend des paramètres à la ligne de commandes en récupère les valeurs à travers les notations `$1`, `$2`,..., `${10}`, `${11}`,... Ces valeurs lui sont passées par le shell appelant.

Dans l'exemple précédent, deux paramètres positionnels ont été utilisés : le paramètre 1 à travers lequel le script récupère une chaîne de caractères et le paramètre 2 à travers lequel il récupère le chemin vers un répertoire.

Attention, on n'affecte pas ces paramètres directement à la main comme on le ferait pour une variable par `var=value`. Ils sont affectés par le shell au moment où le script est lancé.

Les paramètres positionnels peuvent être affectés par la commande interne `set`. Ainsi, une alternative à l'écriture du script précédent pourrait être...

A terminal window titled "Terminal - user1@salle231-05:~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows a sequence of commands and their outputs. The first command is 'ls BBB', which outputs 's t x y z'. The second command is 'set new ./BBB'. The third command is a loop: 'for n in `ls \$2/*` ; do mv \$n \$n.\$1 ; done'. The fourth command is 'ls BBB', which outputs 's.new t.new x.new y.new z.new'. The prompt is currently at the end of the last line.

```
Terminal - user1@salle231-05:~
File Edit View Terminal Tabs Help
[user1@salle231-05 ~]$ ls BBB
s t x y z
[user1@salle231-05 ~]$ set new ./BBB
[user1@salle231-05 ~]$ for n in `ls $2/*` ; do mv $n $n.$1 ; done
[user1@salle231-05 ~]$ ls BBB
s.new t.new x.new y.new z.new
[user1@salle231-05 ~]$
```

➡ Paramètres spéciaux du shell.

On ne considère ici que ceux en lien avec les paramètres positionnels.

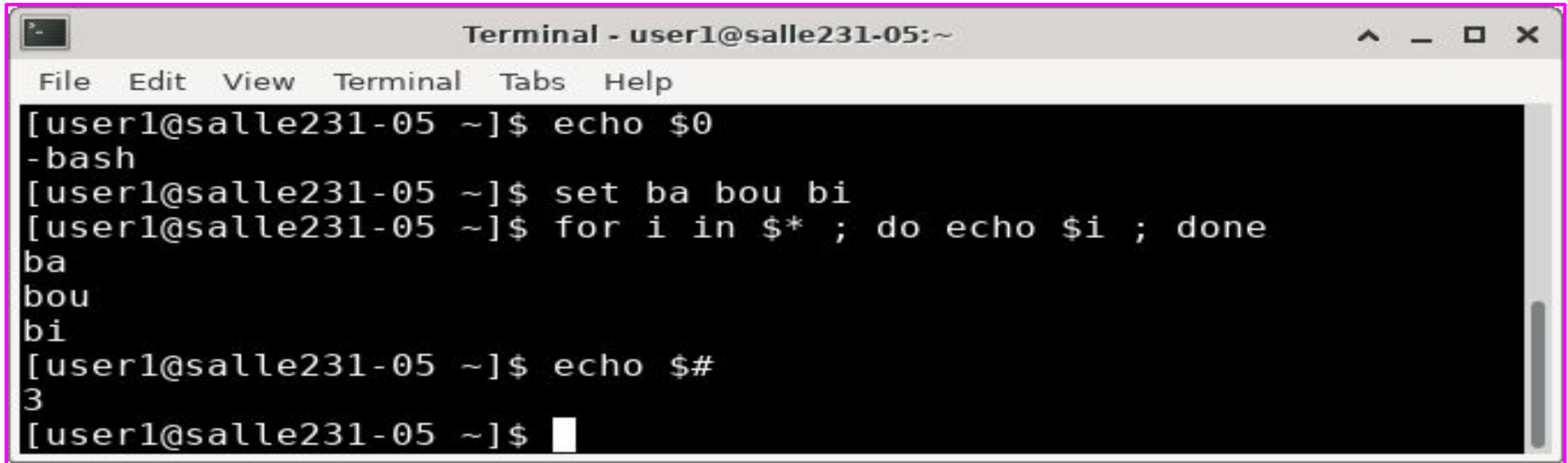
* se développe en les paramètres positionnels en commençant pas 1.

se développe en le nombre de paramètres positionnels en décimal.

0 se développe en le nom du shell ou du script-shell.

L'exemple du script *example.sh* a déjà illustré la signification des paramètres 0 et #

L'exemple suivant montre leur utilisation directement dans le shell (en dehors d'un script-shell).

A terminal window titled "Terminal - user1@salle231-05:~" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows a series of commands and their outputs. The prompt is [user1@salle231-05 ~]. The commands and outputs are: echo \$0 (output: -bash), set ba bou bi (output: ba, bou, bi), for i in \$* ; do echo \$i ; done (output: ba, bou, bi), echo \$# (output: 3), and a final prompt [user1@salle231-05 ~]\$ with a cursor.

```
Terminal - user1@salle231-05:~
File Edit View Terminal Tabs Help
[user1@salle231-05 ~]$ echo $0
-bash
[user1@salle231-05 ~]$ set ba bou bi
[user1@salle231-05 ~]$ for i in $* ; do echo $i ; done
ba
bou
bi
[user1@salle231-05 ~]$ echo $#
3
[user1@salle231-05 ~]$
```

Portabilité des scripts-shell.

Lorsqu'un script-shell est écrit dans une syntaxe de shell autre que celle du shell appelant, il n'est pas garanti que l'exécution puisse se passer correctement.

Que se passe-t-il si on emporte son script-shell écrit en bash et qu'on le lance dans un environnement où le shell est tcsh ?

Afin de garantir la portabilité de son script-shell et sous-réserve que, dans l'environnement où on emporte son script-shell est installé le shell dans lequel est écrit le script, il est fortement recommandé de commencer le script-shell par le préambule `#!/<chemin vers l'exécutable correspondant au shell qui doit interpréter le script>`

Le shell qui va être créé pour exécuter le script-shell va alors se recouvrir par le code exécutable correspondant au shell indiqué après `#!` et en recevant comme argument le nom du script-shell.

```
Terminal - user1@salle231-05:~
File Edit View Terminal Tabs Help
#!/bin/bash
while [[ -z $nothing ]]
do
    echo "ZZZZ"
    sleep 2.5
done
exit
(END)
[0] 0:less+ "user1@salle231-05:~" 08:43 13-oct.-20
```

```
Terminal - user1@salle231-05:~
File Edit View Terminal Tabs Help
[user1@salle231-05 ~]$ ./loop.sh
ZZZZ
ZZZZ
ZZZZ
ZZZZ
ZZZZ
ZZZZ
^C
[user1@salle231-05 ~]$
[0] 0:bash+ "user1@salle231-05:~" 08:47 13-oct.-20
```

```
[user1@salle231-05 ~]$ ps -C "bash loop.sh" -o tty,pid,p
pid,pgid,tpgid,cmd
TT      PID  PPID  PGID  TPGID  CMD
pts/0   12010 12009 12010 12388  -bash
pts/1   12391 12390 12391 12450  -bash
pts/2   12396 12390 12396 12453  -bash
pts/1   12450 12391 12450 12450  /bin/bash ./loop.sh
[user1@salle231-05 ~]$
```